1. INTRODUCTION

A disassembler is a program which, as the name implies, provides a function opposite that of an assembler. An assembler takes text files which you type in, and converts them to object files which can be used by the computer. These object files, while perfect for a computer, are not suitable for human consumption. Thus, the job of a disassembler is to take an object file, and from it create a text file which can be understood. Assemblers and disassemblers are therefore complementary programs.

A disassembler can be useful for many jobs. You might have a program which needs to be modified, but have only an object file of the code. With a disassembler, you can translate this object into a text file, which you can then update as required. The disassembler will not do the whole job, but a good disassembler will make the job infinitely easier. A good way to become a better programmer is to study the programs of others, to encounter new methods of programming. With a disassembler, you can do this with very little trouble. Finally, this disassembler in particular can come in very handy with a specific problem encountered by 6800 users who are upgrading to the 6809, but don't want to leave behind existing programs for the 6800. With the disassembler, those programs can be disassembled, then reassembled under the 6809, after making whatever changes need to be.

Disassembling can be a difficult job, which can be simplified by a powerful disassembler. A simple disassembler may do nothing more than translate object code into absolute source code, without supplying labels to allow easy moving of the program to another part of memory. While adequate for small jobs, this type of disassembler will not suffice for larger jobs, where the existence of labels can be a decided advantage, if not a necessity. Another needed ability of a good disassembler is the ability to specify portions of a program as data instead of code which is meant to be executed by the computer. Such data areas occur as strings of ASCII characters for output or tables of byte or word data for reference by the program. If a disassembler attempts to disassemble these data areas, incorrect code can be produced, or labels can be generated where they need not be. Finally, a useful addition to a disassembler might be the ability to specify that any reference to certain locations should be labeled not by an internally generated label, but by some standard symbol name. Often, a program will reference some location in the operating system which is normally known by a certain standard name. If the disassembler uses this standard name instead of a less meaningful label, a program becomes much easier to understand on first reading.

Obviously, a disassembler cannot perform all of these functions unaided. For example, a disassembler cannot know where all the data areas in a program are located, but it can make it easier to find them. Disassembling a program becomes a cooperative effort between you, the user, and the disassembler. A well designed disassenbler can make this interaction relatively easy, maybe even fun.

DYNAMITE is such a disassembler. It has many abilities not usually found in a disassembler, and can make your disassembly jobs much more agreeable. All of the above mentioned functions are available, as well as many others. A partial list of DYNAMITE's abilities are

-Runs under FLEX 9.0(TM) for the 6809.

-Can disassemble both 6800 and 6809 object files.

-Produces source code which, when reassembled, is identical to the original binary file.

-Automatically produces labels for any memory references within a program. External labels are defined by EQU statements grouped together at the start of the text file.

-Any label can be given a standard name, which is printed instead of the program-generated label name. In the default configuration, references to standard FLEX locations will be given their standard names, i.e. a jump to-the warmstart entry of FLEX will be disassembled as

JMP WARMS

You can define standard names for additional labels, or use different labels in place of the default assignments in a particular program.

-Any block of code within a program can be treated as data. Such an area can be defined as any of four different varieties:

-ASCII data areas will produce FCCs, with strings automatically delimited, and ASCII control characters referred to by their standard names.

-Byte data areas will produce FCBs of hex data bytes.

-Label data areas will produce FDBs of labels references to the program. Many programs have data tables in them which are used for indirect addressing or for performing multi-direction jumps. Such tables are best disassembled as a series of labels.

-Word data areas will produce FDBs of hex data double bytes.

-You can specify the boundaries for data areas within a program in two ways. The data area addresses can be input interactively at the beginning of a disassembly, or can be read from a disk text file which has been set up previously.

-The output from DYNAMITE can be directed to the output device or to a disk text file. The text file created has all extra spaces deleted to save disk space.

-The output listing can be line numbered or paginated.

-To help in finding the data areas in a program, the ASCII equivalent of

---

the code being translated can be printed alongside the source line.

2. TERMS

2.1 FILES

DYNAMITE uses several files to perform disassembly jobs. Initial definitions for these files follow (more complete definitions are given in the instructions):

INPUT FILE is the binary object file which DYNAMITE is directed to disassemble.

OUTPUT FILE is the text file which is optionally created by DYNAMITE to hold the disassembled code for further editing.

COMMAND FILE is an optional text file which contains lines specifying the address boundaries and types of data areas within a program. Also within the command file are the lines specifying the user defined label files (below) to use, as well as lines specifying options normally sent via the FLEX commnand line to specify default options.

LABEL FILE is a binary file which sets up the equivalence between memory addresses and standard symbol names for those addresses. DYNAMITE defaults to using label files which name the standard FLEX locations. The default file used is DISLBL09.BIN for 6809 code, and DISLBLOO.BIN for 6800 code. You can specify a single label file to be used in addition to the default label file, or to be used in place of the default.

2.2 CONVENTIONS

As is standard in most 6809 program documentation, certain conventions will be used in declaring the format of user directions for DYNAMITE. Angle brackets (<>) will-be used to enclose essential elements of a statement. Square brackets ([]) will be used to enclose optional elements.

3. THE FLEX COMMAND LINE

The full syntax for calling DYNAMITE from FLEX is as follows:

+++DYNAMITE <input file> [<output file>] [<options> [+<command file>]

All of the files default to the working drive.

The <input file> is the file from which the object code in read. It has a default extension of .CMD. As normal, another extension or drive may be used by specifying it in the file name. Some examples:

```
+++DYNAMITE PROG                Default to PROG.CMD on working disk

+++DYNAMITE 0.PROG.BIN          Override defaults on filename
```

If the <output file> is not specified, but disk output is enabled, then the output file will default to the name of the input file, with the extension .TXT. The <output file>, if specified, defaults to a .TXT extension. If the output file already exists, then DYNAMITE will ask if the existing file should be deleted:

```
+++DYNAMITE PROG PROGSRC.TXT
output file exists - Delete It (Y-N)?
```

If N is typed, then DYNAMITE will stop and return to FLEX. If Y is typed, the file will be deleted and a new text file created. If anything else is typed, the question will be re-asked.

<options> specify a list of disassembly options on the command line. The list must start with a plus sign ('+'), not have any embedded spaces, and contain the following option switches in any order.

A Print the ASCII equivalent of the code on the source line. These characters will be printed between the code bytes in hex and the label field. Illegal ASCII characters will be replaced by a period in this field.

B Prompt the user for the data area boundaries. This directs DYNAMITE to enter an Interactive mode, requesting data area types and addresses from the user. The prompting format is detailed in the next section.

D Do not create a text file on the disk. The output file will not be created even if a name is specified.

G Do not generate extra lines of code bytes when more than four bytes are in an FCC, FCB, or FDB. Some data area disassembled code may normally require more than one line to list all of the code bytes in hex. If this option is enabled, only one line with the first four code bytes is printed.

L Do not send a disassembly listing to the output device.

N Generate line numbers in the output listing. The line number is a decimal number printed at the start of the source line.

P Enable pagination. You will be prompted for a title, which will be printed at the top of every page. The title can be up to 32 characters long. Any excess over 32 characters will be ignored. If pagination is enabled, the listing output is formatted for 66

line pages, with a heading line, several blank lines, 54 code lines, and a form feed at the bottom of each page.

S Prompt for the segment to disassemble. It is possible to disassemble only a portion of the code, and ignore the rest of the program. This option would be helpful if a large program is to be disassembled, and the resultant text file would be too large for a single disk if the entire source was output. You will be prompted for a beginning and ending address, and the disassembly will take place only between these two addresses inclusive.

Z Disassemble the program for the 6800, not the 6809, which is the default.

0-3 Set the disk drive to read the default label file, DISLBL09.BIN or DISLBL00.BIN, from. Normally, the default label file will be read from the same disk as DYNAMITE was read from. This is done by retrieving the disk drive number from the system file control block on entry, and assuming that DYNAMITE was loaded using this FCB. If this is not true, or the default file should be read from a different disk than that holding DYNAMITE, a 0, 1, 2, or 3 in the option list will set the drive number to override the default.

The final element of the command line is the <command file>. This is the file holding the predefined data area boundaries and other miscellaneous commands. The format of this file is described in section 5. The command file has a default extension of .TXT. If specified, the file name must be separated from the rest of the command line by a plus sign ('+'). This is not the same plus sign as that specifying the status of the option list. If no options are specified, two plus signs must still be in the command line. For example:

```
+++DYNAMIITE OBJPROG ++CMDFILE
```

What follows are several examples of calls to DYNAMITE, with explanations of the effect of each call:

```
+++DYNAMITE PROGA +DAN
```

Disassemble program PROGA.CMD, from the working disk. Do not create any disk file, but do list the disassembly. The listing should be line-numbered, and have the ASCII equivalent of the code printed in the code line. This option list is normally used to determine where in a program the data areas, especially the ASCII strings, are located, so that a more accurate disassembly can be made.

```
+++ DYNAMITE OBJFILE.BIN +DNGB
```

Disassemble program OBJFILE.BIN, read from the working disk. Do not create a disk file. The output listing should have line numbers, and FDBs, FCBs, and FCCs should not run to extra lines. Before the disassembly starts, prompt the user for the addresses of data areas

within the program.

```
+++DYNAMITE TESTFILE.0 1.SRCFILE +ZLS
```

Disassemble program 0.TESTFILE.CMD, and create a text file 1.SRCFILE.TXT for the source text. TESTFILE should be disassembled as a 6800 program, not 6809. Do not print an output listing, and before disassembly begins prompt the user for addresses between which the program should be disassembled.

```
+++DYNAMITE FILEA FILEZ +PGN1 +FILEABND
```

Disassemble the program FILEA.CMD from the working drive, and store the text output in the file FILEZ.TXT on the working drive. The output listing should be paginated and have line numbers, and extra lines of code generation should be suppressed. Before disassembly starts, the command file FILEABND.TXT from the working drive should be processed. The default label file, DISLBL09.BIN, should be read from drive 1, regardless of the disk that DYNAMITE was read from.

4. USER INPUT BEFORE THE DISASSEMBLY

Three options, the B, P, and S options, require user input before the disassembly can begin. The format of that input will be described below.

4.1 THE TITLE INPUT

The first user input, if pagination is enabled, will be the title to be printed at the top of every page. DYNAMITE will print the prompt

```
Title?
```

and wait for input from you. Now type in the title you want, then press RETURN. The title can be at most 32 characters long, so any excess typed in is simply ignored.

4.2 THE DATA AREA BOUNDS INPUT

The next user input is that required by the B option. DYNAMITE will enter an interactive mode enabling you to specify the addresses of data areas within the program to disassemble. DYNAMITE will print the prompt

```
Data Segment Type: Ascii, Byte, Label, Word, Reset, or Proceed?
```

and wait for your response. Now enter the command for the type of the text data area. Only the first letter of the type must be given, as that

is all that is checked. The meaning of the commands in the type prompt are:

A This declares a data area to be ASCII, so that it will be disassembled as FCCs. ASCII control characters will be generated using their standard names, with EQU statements collected at the start of the program. Other unprintable ASCII characters will be printed as two digit hex bytes, preceded by a dollar sign '$'. For the 6809, all of these elements will be generated together in FCCs, separated by commas, since this syntax is allowed by the TSC Mnemonic Assembler. For the 6800, though, this format is illegal, so unprintable ASCII characters will be generated on FCB lines within the FCC data area.

B This declares a data area to be byte data, so that it will be disassembled as FCBs. Each byte in the data area will be printed with two hex digits preceded with a dollar sign. A maximum of eight bytes will be generated in each FCB.

L This declares a data area to be label data, so that it will be disassembled as FDBs of labels. Each double byte in the data area will be printed as a label, and that label will be defined elsewhere in the output. This is useful for jump tables or tables of indirect addresses.

W This declares a data area to be word data, so that it will be disassembled as FDBs of constants. Each double byte in the data area will be printed as four hex digits preceded with a dollar sign. For both the label and word segments, a maximum of four words will be generated in each FDB.

R This does not specify a data area type, but instead directs DYNAMITE to erase all data area specifications input so far, and start over on the bounds input from scratch. Due to the structure of the data area bounds processing, it is not possible to back up a single step in the prompting sequence, so the only recourse if an error is made is to reset everything to the start and reenter all the data area bounds.

P This directs DYNAMITE to proceed from the prompting sequence to the next action. In other words, stop prompting for data area addresses and continue with the disassembly.

Any illegal type character will be ignored, and the prompt reissued.

The A, B, L, and W type characters require a pair of addresses to be input. These are the first and last address in the data area. DYNAMITE will first prompt with

Starting address?

to which you should respond with the hex address of the start of the data area. Next, DYNAMITE prompts

Ending address?

to which you should respond with the hex address of the end of the data area.

If the type character entered was not P, Proceed, then the prompt will be reissued so that the next data area can be specified.

4.3 PROMPTING FOR THE SEGMENT TO DISASSEMBLE

If the text of a disassembly would be too large to save on a single disk, it can be split up using the S option. If the S option is specified, then DYNAMITE will prompt with

What are the bounds of the segment to disassemble?

DYNAMITE then prompts for the first and last address of the segment to disassemble, in exactly the same way as the address pairs are prompted for while inputting the data area bounds.

5. THE COMMAND FILE

While you can always specify the addresses of a program's data areas by invoking the B option, this requires that you enter all of the data area bounds every time you disassemble the program. This is acceptable for small programs, but can be a substantial job for any large programs. The entry of the specifications may take several minutes each time, and a single error would require starting over from scratch. To make your job easier, DYNAMITE has the ability to read the data area specifications from a text file, as well as from the terminal at run time. This is the main function of the command file. You also use the command file to specify the name of a label file,which is to be used in addition to the default label file, or as a replacement for the default. Finally, the command file can also be used to specify options so that they need not be declared in the FLEX command line for every disassembly.

The command file should be created as a series of command lines, each holding a single command. The command character should be the first character in the line, and be followed by whatever parameters the line requires. The various commands which are accepted within the command file are:

+ <options>

This command line specifies a list of options, whose allowable values are identical in meaning to the options specified in the FLEX command line. This command is used for options which would always be specified for a program being disassembled. For example, a common usage of the options command would be to specify a program as 6800 code, so that you do not need to remember to do this on

every disassembly. The command in this case would be '+Z'. All of the options for the FLEX command line are legal within the command file except for the B, D, P, and S options. These are not allowed because, for the D command, any disk option file has already been opened before the command file is processed, and for the B, P, and S options, all user input has been requested before the command file is processed.

A <start addr>-<end addr>

This command line specifies the addresses of a data area to be disassembled as FCCs or ASCII data. As such, its effect is the same as the A type character entered in response to the B option. <start addr> and <end addr> are the hex values of the first and last addresses of the data area being specified, with the first address less than or equal to the last. The addresses should be separated by a dash, '-', without any spaces between them. For example, to declare the zero page of memory as an ASCII data area, the commands 'A 0-FF' or 'A 0000-00FF', would be used.

B <start addr>-<end addr>

This command line specifies the addresses of a data area to be disassembled as byte data in FCBS. Its result is the same as the B command in the user data area specification. The parameters take the same format as for the A command line.

L <start addr>-<end addr>

This command line specifies the addresses of a data area to be disassembled as FDBs of labels. Its result is the same as the L command in the user data area specification. The parameters take the same format as for the A command line.

W <start addr>-<end addr>

This command line specifies the addresses of a data area to be disassembled as FDBs of words, or double bytes. Its result is the same as the W command in the user data area specification. The parameters take the same format as for the A command line.

S <label file>

This command line specifies the name of a label file to be used in place of the default label files, DISLBLO9.BIN or DISLBL00.BIN. The default label file is always used if this command is not issued. The file name should have the standard FLEX file name format. The file will default to the working drive, with a default extension of .BIN. A special form of the S command is simply 'S', with no file specified. If such a command is found, then no label files will be used, and all labels in the disassembly will be assigned program-generated names.

T <label file>

This command line specifies the name of a label file to be used in addition to the default label file. The file name must be in the standard FLEX format, and defaults to the working drive with an extension of BIN.

The +, A, B, L, and W command lines may occur as many times as is necessary within the command file, and do not have to be in any particular order. Only one label file can be used in place of or in addition to the default label file, so only a single S or T command may be used. The command file can only have one S command, one T command, or neither. You cannot specify both the S and T options within a single command file.

You may also include blank lines or lines beginning with an asterisk, '*', in the command file wherever desired. These lines are ignored by the command file processor, and can be used as comment lines or to format the command file for easy viewing and editing.

As an example of a command file, consider the effect of the following command lines:

```
+ZG
*  Data areas within the code
B A102-A104
W A105-Al08
L A233-A240
A A241-A27F
* Use an additional label file
T LBLFIL.BIN
```

The first command line declares the program being disassembled to be 6800 code, and directs DYNAMITE to suppress extra generation of code byte lines for FCCs, FDBs, and FCBS. The second line, starting with an asterisk, is a comment line, ignored by DYNAMITE. The third through sixth lines declare four data areas within the program, one of each data area type. The code from A102 to A104 should be disassembled as bytes in FCBs. The bytes from A105 to A108 should be disassembled as two words in FDBs. The bytes from A233 to A240 should be disassembled as seven labels of FDBs. Finally, the bytes,from A241 to A27F should be disassembled as ASCII data, in ASCII strings or as ASCII constants. The seventh line is another comment line, and is ignored. The eigth line specifies a label file, LBLFIL.BIN, to be read from the working drive and used in addition to the default label file.

6. THE LABEL FILE

Normally, you don't have to worry about the assignment of standard names to predefined memory addresses. This function is automatically taken care of by DYNAMITE, which will refer to FLEX addresses by their normal names. You may want to expand the scope of the assignments, though, either for a single program or for all disassemblies. For a single program, this involves creating a custom label file for the program, and placing the name in an S or T command within the command file. To change the equivalences for all disassemblies, you will have to modify the default label files, DISLBLO9.BIN or DISLBL00.BIN. To accomplish either function, you must know the format of the label files.

DYNAMITE label files must be binary object files, with code produced by an assembler. To create a label file, you must first type in an assembler source file, from which the binary file is produced. The label source file should be a series of pairs of FCCs and FDBs. The FCC in a pair specifies the standard name for a location, while the FDB specifies the location value. The name FCC must assemble as six data bytes. If the name is less than six bytes long, it should left justified in the FCC with spaces padding the unused name bytes. The FDB should specify a single word address. The last entry in the file should be a 'name' of six spaces, that is, an empty name, followed by an FDB specifying a zero address. As an example, consider a small label file, assigning the name 'VN' to address C102, the name 'STARTL' to address C100, and the name 'MEMHI' to address FFFF. The assembler source file would be set up as follows:

```
        FCC   "VN    "
        FDB   $Cl02
        FCC   "STARTL"
        FDB   $C100
        FCC   "MEMHI "
        FDB   $FFFF
        FCC   "      "
        FDB   $0000
```

Remember when creating the label source file that all name FCCs must be six characters long, and that the list of equivalences must be terminated with a name of six spaces.

Once the label source file is created, it must be assembled to create the binary label file. The name of the label file can then be used within command files for any disassemblies.

The disk on which DYNAMITE has been sent to you also contains the source files from which the default label files were created, DISLBL09.TXT and DISLBL00.TXT. You can study these files to better understand how label files should be set up. If desired, you can modify these files, so that a different set of default standard label names can be used. You will also find three other source label files, along with the binary label files themselves. MFLEXLBL.TXT is the label source file of label assignments for MINIFLEX(TM), a predecessor of the FLEX operating system on the 6800. SWTBGLBL.TXT is the label source file of label assignments for SWTBUG(TM), the system monitor in SWTPC 6809 computer systems. SBUGLBL.TXT is the label source file of label assignments for S-BUG(TM), the system monitor in SWTPC 6809 computer systems. These three label files, if needed, can be used in three ways. First, the text files can be used as starting points for custom label files that you create. Second, the binary label files can be appended onto existing label file segments to create a new label file. Note that if this is done, the label file segment cannot have the terminating entry of a blank name, and the standard label file you are using must be appended last, since all are shipped with the terminating entry supplied

---

within them . Finally, either standard label file may be used as is, since all have the correct format for label files.

7. DISASSEMBLY PARTICULARS

There are several particular attributes of DYNAMITE which are useful to keep in mind when disassembling.

DYNAMITE is a three pass disassembler. On the first pass, the object code is searched for memory references, which are to be translated as labels. As well as the obvious memory references, such as the direct or extended addressing modes, DYNAMITE regards the immediate addressing mode using the IX, IY, U, or S registers as memory references. Thus, a LDX # (load IX immediate) encountered within the object code will generate a label for the immediate value. This ensures that the vast majority of labels will be picked up. Unfortunately, this also means that an immediate load of those registers with data will also generate labels, when this was not what was desired. For instance, in some code, IX is loaded with a constant to control a loop count, as in LDX #256. This will be disassembled as LDX #L0100, where L0100 is the label assigned a reference to address 256, $100 in hex. Thus, there will usually be some labels generated which are actually data values instead of memory locations. An exception to the generation of labels from immediate addressing is in the immediate mode with ACCD, as in ADDD #1. Since ACCD is almost exclusively used for data manipulation instead of addressing, immediate references are always disassembled as data. While this may, very rarely, miss a label which should have been generated, it will prevent the generation of many unneeded labels.

The second pass of DYNAMITE does not attempt to disassemble the code but simply flags all labels found in pass 1 as internal or external. Internal labels are those whose value is the address of code within the program, while external labels are those not occuring ,within the program as code addresses. This pass is required so that all external labels can be defined using equate statements at the beginning of the disassembly listing.

The third pass of DYNAMITE creates the actual disassembly code, both the disk file and the output listing. The output has four main parts. First, all standard labels, whose names were read from the label files, were referenced within the code, and are external to the program, are grouped together and defined using equates. Second, all ASCII equates are grouped together. If any ASCII data areas were defined, and control characters were encountered within these areas, then the standard names for the control characters are used in the FCC statements. These standard control character names are defined in this section of the output. Third, all external labels without standard names are defined by equates. This output section should be studied carefully, because any generated labels which should really be data will normally show up in this equate segment. If a label is defined which does not appear to be a true legal memory reference, check in the actual disassembly for all

references to the,label and determine for yourself if it should actually be a data value. The fourth output section is the main section, the actual disassembly. This will be a series of assembler-compatible source lines. If any lines are referenced in the code, then the internally generated label will be placed in the label field of the source line. All label references appear either as a standard name, if the value was found in a label file, or as an L followed by a four digit hex number, whose value is the reference address. Thus, references to address $1234 will generate a label of L1234.

The code produced by DYNAMITE should always reassemble into code which is identical with the original source file. Some addressing modes, though, can generate either 8 bit references or 16 bit references, especially on the 6809. To ensure that such references will assemble to code with the same reference bit length, DYNAMITE will generate forced addressing where required. This can be done only for the 6809, and is compatible with the forced addressing method expected by TSC's Assembler. Thus, all direct references, as well as all 8 bit PCR relative addressing will have the 'force 8 bit addressing' character, the less than sign, '<', before the label. For example, a direct load into ACCA from address $55 will be disassembled as LDA <L0055. Also, all 16 bit references for which 8 bits suffice will have the 'force 16 bit addressing' character, the greater than sign, '>', before the label. An extended store of IX to address $33 will be disassembled as STX >L0033.

8. A DISASSEMBLY EXAMPLE

To help you understand the procedure for disassembling a program, a fairly routine example which uses many of DYNAMITE's abilities will be discussed. The program to be disassembled will be written in 6800 assembly language, and will be converted, using DYNAMITE, to run under the 6809 with FLEX 9.

The process of disassembling a program with DYNAMITE is usually a multi-step procedure. First, you need to determine where the data areas occur within a program. To do this, disassemble the program, suppressing disk output and generating the ASCII equivalent of code in the listing. By studying this listing, all of the ASCII strings can be easily found, and some idea of the location of other types of data areas can be formed. For instance, most programs running under FLEX will start with a branch instruction over the version number and other data variable areas. Such an area can be defined as some combination of byte and word length data areas. Once an idea is formed of where the data areas occur, the program can be disassembled again, this time with the D and B options, so disk output is again suppressed, but this time prompting you for the data areas discovered in the first disassembly. This disassembly will be much closer to the final version, and any data areas missed the first time can usually be found now. Finally, you can create a command file with the final version of the data area boundaries, and disassemble the program once more, this time using the command file and creating a disk text file.

If you are converting a program from 6800 to 6809, the next step would be to edit the text file. Since all external labels are grouped together, it is a simple matter to update all FLEX 2.0 references at the start of the program to FLEX 9.0 addresses, usually just by changing every $A to $C, $B to $D. Also, find all of the ORG statements within the text, and update theg as needed. Once this is done, simply assemble the program using the 6809 assembler. Your conversion is complete.

The example disassembly will follow this method. All of the files referred to below exist on the disk on which DYNAMITE was shipped, so follow along.

The first step, as outlined above, is to disassemble the program with ASCII generated, to find the data areas. The command to do this is (assuming that DYNAMITE and the standard label files are on the system drive, TESTFILE.BIN on the working drive):

```
+++DYNAMITE TESTFILE.BIN +DANZ
```

This will disassemble the program for the 6800, suppress disk output, and add line numbers and ASCII code equivalents to the listing. The listing produced can be found on the next page.

```
 1                                * DISASSEMBLY BY DYNAMITE OF 1.TESTFILE.BIN
 2
 3                                * STANDARD PRE-NAMED LABEL EQUATES
 4
 5                        AD03  WARMS   EQU    $AD03
 6                        AD1E  PSTRNG  EQU    $AD1E
 7
 8                                * EXTERNAL LABEL EQUATES
 9
10                        A16E  LA16E   EQU    $A16E
11
12  A100                                 ORG    $A100
13  A100 20   04           .     LA100   BRA    LA106
14  A102 01               .              NOP
15  A103 00               .     LA103   FCB    $00
16  A104 00               .              FCB    $00
17  A105 00               .              FCB    $00
18  A106 B7   A103        ...   LA106   STAA   LA103
19  A109 BF   A106        ...           STX    LA104
20  A10C CE   A115        ...           LDX    #LA115
21  A10F BD   AD1E        ...           JSR    PSTRNG
22  A112 7E   AD03        ~..           JMP    WARMS
23  A115 54               T     LA115   LSRB
24  A116 45               E             FCB    $45
25  A117 53               S             COMB
26  A118 54               T             LSRB
27  A119 20   53           S            BRA    LA16E
28  A11B 54               T             LSRB
29  A11C 52               R             FCB    $52
30  A11D 49               I             ROLA
31  A11E 4E               N             FCB    $4E
32  A11F 47               G             ASRA
33  A120 04               .             FCB    $04
34                         A100          END    LA100
```

By scanning the ASCII equivalent area of the listing, It is obvious that the code from $A115 to $A120 is an ASCII string. The $04 at $A120 is an ASCII EOT, used by FLEX to terminate ASCII strings, and as such should be included in the ASCII area. Next, most FLEX programs begin with a branch past a data area, so the BRA LA106 at $A100 implies that $A102 to $A105 should be either byte or word data. Since the instruction at $A109 saves a 16 bit register, X, at LA104, you can assume that $A104 to $A105 should be a word type data area. Also, the byte following the branch, the 1 at $A102, is normally the program version number, so you can assume that $A102 to $A103 should be a byte type data area.

The remainder of the program seems to have disassembled properly, so all of the data areas have probably been found. Note the external label LA16E that has been generated. It is referenced only in the code that should be an ASCII string, so that when we enter the data areas to DYNAMITE, there should be no external labels.

Now, you can proceed with the second disassembly, this time entering the data boundaries to DYNAMITE. Since there is little chance that a third disassembly will be required, the output can be written to disk at this time. The command sent to FLEX is now:

```
+++DYNAMITE TESTFILE.BIN +BNGZ
```

This command will have DYNAMITE input, via the keyboard, the data area addresses found earlier. Also, the output listing will be line numbered, and extra lines for the ASCII string will not be generated. The program will again be treated as 6800 code. Note that the ASCII equivalent area of the listing is turned off, since it is no longer required.

After DYNAMITE has begun execution, the prompt for the entry of the first data area type will be sent. You want to enter the area from $A102 to $A103 as byte data, so type a 'B', then RETURN. DYNAMITE will ask for the starting address of the data area. Type the characters 'A102', then RETURN. The dollar sign is not required, since hex is assumed. Next, DYNAMITE will ask for the ending address of the data area. To this prompt, answer 'A103', then RETURN.

DYNAMITE reissues the prompt for data area type. You should now specify a word type data area from $A104 to $A105. Type 'W' in response to the first question, 'A104' to the second, and 'A105' to the third. In the same way, specify an ASCII type data area from $A115 to $A120. After this, all of the data areas should have been entered. If at any point you made a mistake which DYNAMITE did not immediately flag as an error, type an 'R' to the data area type prompt and start over. Otherwise, the data area specification is done. Type a 'P' to indicate that DYNAMITE should Proceed to the rest of the disassembly process.

DYNAMITE will finish up its initial processing before beginning its three passes. At the start of pass one, a RETURN, LINE FEED is sent, and disassembly begins. After pass three is complete, the output listing produced should be the same as the listing on the next page.

```
 1                      * DISASSEMBLY BY DYNAMITE OF 1.TESTFILE.BIN
 2
 3                      * STANDARD PRE-NAMED LABEL EQUATES
 4
 5                 AD03 WARMS   EQU    $AD03
 6                 AD1E PSTRNG  EQU    $AD1E
 7
 8                      * ASCII CODE EQUATES
 9
10                 0004 EOT     EQU    $04
11
12
13  A100                        ORG    $A100
14
15  A100 20   04       LA100    BRA    LA106
16  A102 01                     FCB    $01
17  A103 00            LA103    FCB    $00
18  A104 0000                   FDB    $0000
19  A106 B7   A103     LA106    STAA   LA103
20  A109 BF   A106              STS    LA106
21  A10C 8E   A115              LDS    #LA115
22  A10F BD   AD1E              JSR    PSTRNG
23  A112 7E   AD03              JMP    WARMS
24  A115 54 45 53 54   LA115    FCC    "TEST STRING"
25  A120 04                     FCB    EOT
26
27                 A100         END    LA100
```

This latest disassembly is the final one required. All of the data areas are formatted correctly, and the output on disk is ready for editing so it can be updated to run on the 6809. Thus, you next edit the program TESTFILE.TXT.

The first thing to be done in fixing the text is to update the addresses of the FLEX variables from those used by FLEX 2 to those of FLEX 9. Thus, the equate addresses given for WARMS and PSTRNG should be changed from $AD03 and $AD13 to $CD03 and $CD13 respectively. Also, the origin address for the program should be changed from $A100 to $C100, moving it into the FLEX 9 command file buffer. This is all the editing which must be done, but you may chose to change all the labels from LAxxx to LCxxx, so that they reflect the actual, new addresses, not the old ones. This is not necessary, though, and may not always help much since, because 6800 and 6809 source code lines do not always translate to the same number of bytes, the labels may still not reflect the correct addresses.

Once the editing is complete, assemble the file TESTFILE.TXT. If using the TSC Assembler, the command might be

```
+++ASMB TESTFILE TESTFILE.CMD +GN
```

or any other such line creating a 6809 type CMD file. Now execute the new command file, which should simply type the ASCII string 'TEST STRING' and return to FLEX. The conversion of a 6800 program to the 6809 is complete.

While other programs to be disassembled will not be as simple as the example program, the procedure remains the same. Thus, as long as you are careful in finding the data areas within a program, practically any program can be disassembled and updated to run as you wish.

As a further example, listed below are the lines which would be placed in a text command file so that the prompting for data areas need not be done at DYNAMITE run time. The command file would have the lines:

```
+ZG
B A102-A103
W A104-A105
A A115-A120
```

This specifies that the program being disassembled contains 6800 object code, with data areas of type and location as determined earlier. If the command file has been called TESTCMD.TXT (as it has on the DYNAMITE disk), then the command to disassemble TESTFILE using the command file would be

```
+++DYNAMITE TESTFILE.BIN +DN +TESTCMD
```

The options typed on the FLEX command line specify that no disk output is to be produced, and that the listing should have line numbers. Note that it is not necessary to declare the code as 6800, or suppress the generation of extra code lines, since the first line of the command file

accomplishes this automatically.

9. ADAPTING TO YOUR SYSTEM

DYNAMITE has been written to run under FLEX 9, and has very little which needs to be modified to adapt it for a particular system. There are, however, two constants available for you to change:

1) The number of code lines printed per page
2) The first illegal ASCII character past $20

The first constant is located at address $003A. This is the number of code lines to be printed on each page of the output listing when pagination is enabled. This number does not include the lines required for the page heading and top blank lines (5) or blank lines at the botton of the page (determined by the physical page size). The skip to the next page is done by outputting a form feed character, ASCII $0C, to the output device. Presently, the constant for the number of code lines per page is set to 54.

The second constant is located at $003B. This number is the value of the first ASCII character past $20 which should not be printed. This value is referenced when printing the ASCII equivalent area of the listing, and when printing ASCII strings generated in ASCII data areas. If an ASCII character to be sent is greater than or equal to this value, a period will be placed in the ASCII area, and the hex value of the character will be placed in the FCC, or FCB for the 6800. This can be useful for certain CRT terminals, which will not accept some ASCII characters normally accepted by other terminals. For instance, Hazeltine terminals, such as the 1500, use ASCII $7E as the escape sequence character, so that $7E is never printed. If an attempt is made to print this as a character, the output will not appear correct. To change DYNAMITE to accomodate the terminal, the constant at $003B should be changed to $7E. The default value, as shipped, is $7F, so that deletes are not treated as printable.

To change either constant, do the following:
First, using the FLEX MAP utility, determine the addresses at which DYNAMITE loads. This will be from $0000 to some number around $2000. Next, load DYNAMITE using the FLEX GET command, and enter the system monitor with the MON command. Change the constants at $003A and $0033 as required. Go back into FLEX via the warm start address, $CD03. Finally save the new version of DYNAMITE on the disk using the FLEX SAVE command. For the start and end addresses of the program, use the values determined by MAP in the first step. Remember to specify a transfer address of 0000.

1. DYNAMITE ERROR MESSAGES

There are two varieties of errors detected by DYNAMITE. The first of these is disk errors, as returned by FMS. The second is non-disk errors, caused by some form of illegal user input detected by DYNAMITE.

Disk errors are at least two lines long. The first line gives the particular file involved. The line has the form

Type File Error

where 'Type' is one of Input, Output, Command, or Label. The last line of the error is the output line returned by RPTERR in FLEX. Thus, if the file ERRORS.SYS is on the system disk, then the error message will be a complete text line, but if the file is not found, a line 'DISK ERROR 'nn' will be printed.

Non-disk errors print error messages found internal to DYNAMITE. These errors give a description of the condition causing the program to be aborted or interrupted. All non-disk errors are detected in the option processing portion of DYNAMITE, before the three passes begin.

Both types of errors may occur while the command file is being processed. If an error occurs in the command file, then another line is included in the error output. This line has the form

Command file line #nnn

where nnn is the line number within the command file on which the error occurred. This line is the second line in a disk error, the first in a non-disk error.

Disk errors always cause DYNAMITE to stop execution. All files are closed, then control is returned to FLEX via the warm start vector. Non-disk errors normally result in the program being halted in the same way. If the error is caused by illegal input during the user input portion of DYNAMITE, though, the error message is not usually a terminating one, but just a request for re-input of the unaccepted data.

The non-disk errors detected by DYNAMITE are:

Syntax error in command line
: Some unrecognizable character sequence was found in the FLEX command line calling DYNAMITE.

Start > End, Reenter both addresses
: When entering a pair of addresses, either data area boundaries or the segment to disassemble, the starting address was greater than the ending address. The input is ignored, and you will be re-prompted for both addresses.

Illegal entry, re-enter
: An illegal hex number was typed. Retype the input.

Command syntax error
: A line was found in the command file which could not be recognized as a legal command line.

Illegal segment address specification
: The starting-ending address pair in a data area address command line was illegal in some way.

Multiple 'S' or 'T' commands in command file
: The command file had either two 'S' commands, two 'T' commands, or at least one of each. Only one 'S' command or one 'T' command is allowed in the command file.

Illegal option switch
: An illegal option character was encountered in the option list in the FLEX command line or in an option command within the command file.

Word or Label segment has odd length
: The addresses specified for a data area of type word or label was an odd number of bytes long. Double byte data areas must have an even length. If this error occurred during user prompting of data areas, the input is ignored. In the command file, this error halts DYNAMITE.

Data segments overlap
: The latest data area occupied some of the same addresses as a previously specified data area. Data areas must be mutually exclusive, in that any address can be in at most one data area. If this error occurred during user prompting of data areas, the input is ignored. In the command file, this error halts DYNAMITE.

Tables out of memory
: DYNAMITE has insufficient memory to build all of the required tables.

This manual his been prepared using the TSC 6809 Text Processor and printed by a XEROX model 1750 daisy-wheel printer.